

Отчёт по лабораторной работе №14

Именованные каналы

Щербак Маргарита Романовна

2022

1 Цель работы:

Приобрести практические навыки работы с именованными каналами.

2 Теоретическое введение:

Одним из видов взаимодействия между процессами в операционных системах является обмен сообщениями. Под сообщением понимается последовательность байтов, передаваемая от одного процесса другому.

В операционных системах типа UNIX есть 3 вида межпроцессорных взаимодействий:

общееюниксные (именованные каналы, сигналы), System V Interface Definition (SVID — разделяемая память, очередь сообщений, семафоры) и BSD (сокеты).

Для передачи данных между неродственными процессами можно использовать механизм именованных каналов (named pipes). Данные передаются по принципу FIFO (First In First Out) (первым записан — первым прочитан), поэтому они называются также FIFO pipes или просто FIFO. Именованные каналы отличаются от неименованных наличием идентификатора канала, который представлен как специальный файл (соответственно имя именованного канала — это имя файла). Поскольку файл находится на локальной файловой системе, данное IPC используется внутри одной системы.

Файлы именованных каналов создаются функцией `mkfifo(3)`.

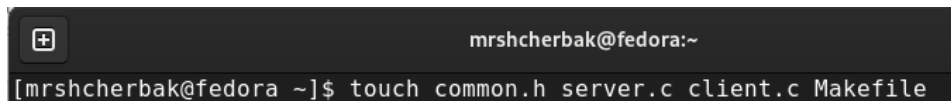
Первый параметр — имя файла, идентифицирующего канал, второй параметр — маска прав доступа к файлу.

После создания файла канала процессы, участвующие в обмене данными, должны открыть этот файл либо для записи, либо для чтения. При закрытии файла сам канал продолжает существовать. Для того чтобы закрыть сам канал, нужно удалить его файл, например с помощью вызова `unlink(2)`.

Каналы представляют собой простое и удобное средство передачи данных, которое, однако, подходит не во всех ситуациях. Например, с помощью каналов довольно трудно организовать обмен асинхронными сообщениями между процессами.

3 Выполнение лабораторной работы:

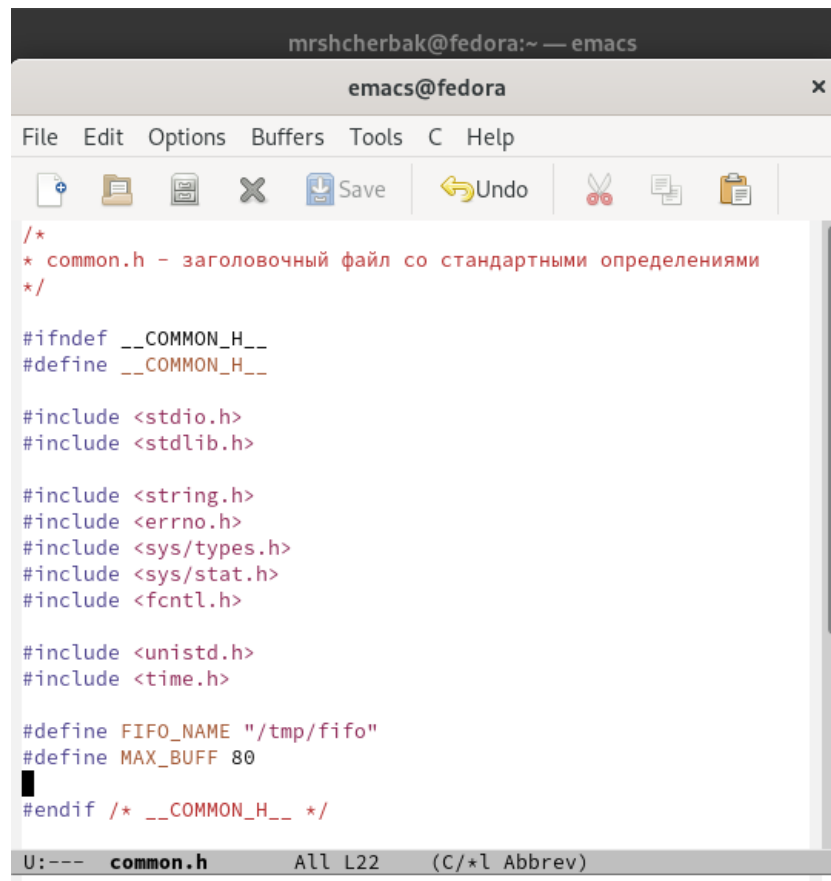
1. Я создала необходимые файлы с помощью команды touch и открыла текстовый редактор emacs для редактирования этих файлов.(Рис. 3.1).

A terminal window with a dark background. The title bar shows a window icon and the text 'mrshcherbak@fedora:~'. The command prompt shows '[mrshcherbak@fedora ~]\$ touch common.h server.c client.c Makefile'.

```
mrshcherbak@fedora:~  
[mrshcherbak@fedora ~]$ touch common.h server.c client.c Makefile
```

Рис. 3.1: Создание нужных файлов

2. Изменила содержимое файлов в редакторе. В файл common.h добавила заголовочные файлы unisd.h и time.h. (Рис. 3.2 - Рис. 3.5).



```
/*
 * common.h - заголовочный файл со стандартными определениями
 */

#ifndef __COMMON_H__
#define __COMMON_H__

#include <stdio.h>
#include <stdlib.h>

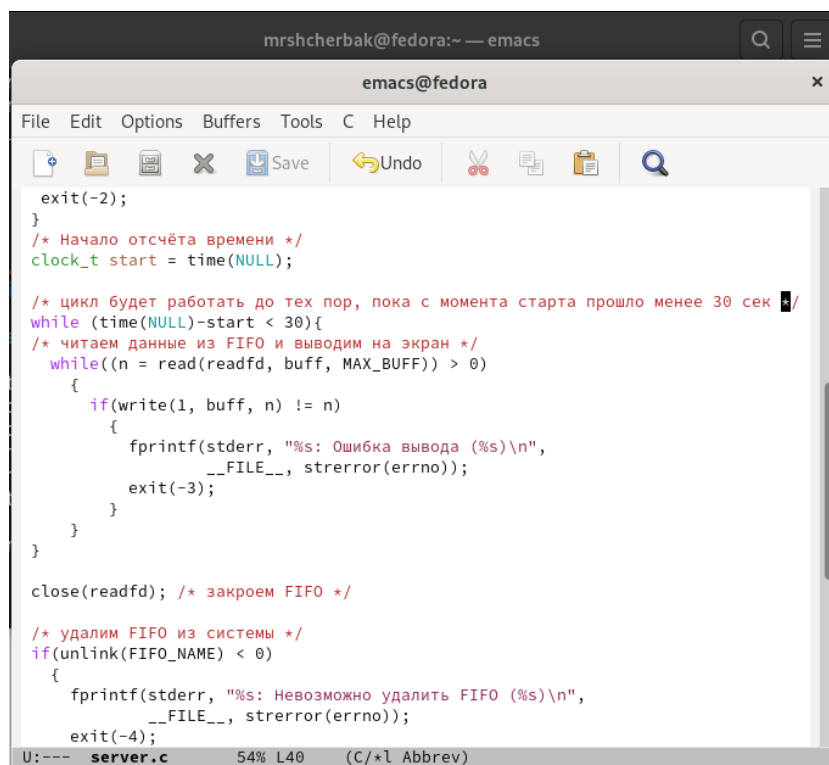
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#include <unistd.h>
#include <time.h>

#define FIFO_NAME "/tmp/fifo"
#define MAX_BUFF 80
#endif /* __COMMON_H__ */
```

Рис. 3.2: Редактируем файл common.h

В файле server.c добавила цикл while для контроля за временем работы сервера.



```
exit(-2);
}
/* Начало отсчёта времени */
clock_t start = time(NULL);

/* цикл будет работать до тех пор, пока с момента старта прошло менее 30 сек */
while (time(NULL)-start < 30){
/* читаем данные из FIFO и выводим на экран */
while((n = read(readfd, buff, MAX_BUFF)) > 0)
{
    if(write(1, buff, n) != n)
    {
        fprintf(stderr, "%s: Ошибка вывода (%s)\n",
            __FILE__, strerror(errno));
        exit(-3);
    }
}
}

close(readfd); /* закроем FIFO */

/* удалим FIFO из системы */
if(unlink(FIFO_NAME) < 0)
{
    fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n",
        __FILE__, strerror(errno));
    exit(-4);
}
```

U:--- server.c 54% L40 (C/*l Abbrev)

Рис. 3.3: Редактируем файл server.c

В файл client.c добавила цикл, отвечающий за кол-во сообщений о текущем времени (4 смс), и команду sleep(5) для приостановки работы клиента на 5 сек.

```
/* цикл, отвечающий за отправку сообщения о текущем времени */
for (int i=0; i<4; i++)
{
    /* получим доступ к FIFO */

    if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
    {
        fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
            __FILE__, strerror(errno));
        exit(-1);
        break;
    }

    /* текущее время */
    long int ttime=time(NULL);
    char *text=ctime(&ttime);

    /* передадим сообщение серверу */
    msglen = strlen(MESSAGE);
    if(write(writefd, MESSAGE, msglen) != msglen)
    {
        fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
            __FILE__, strerror(errno));
        exit(-2);
    }
    sleep(5);
}
```

Рис. 3.4: Редактируем файл client.c

Makefile оставила неизменённым.

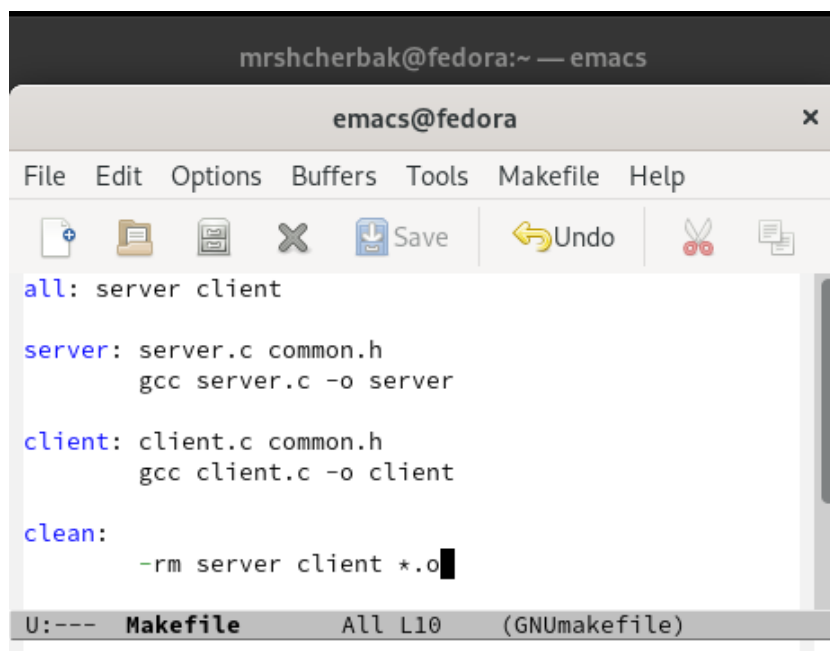


Рис. 3.5: Makefile

3. Скомпилировала файлы (Рис. 3.6).

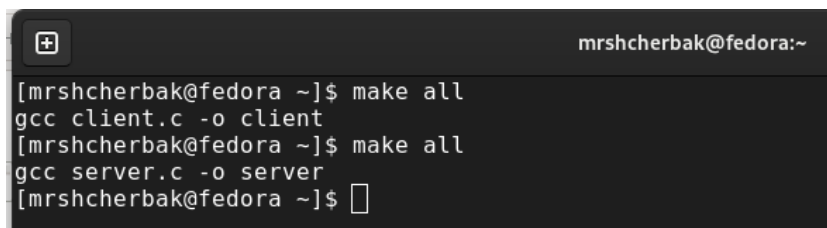
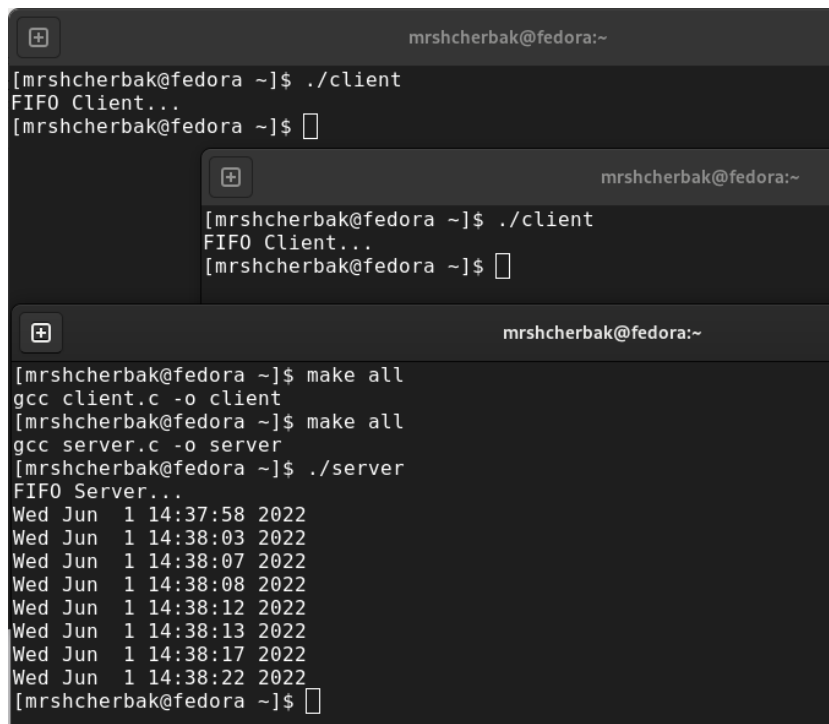


Рис. 3.6: Компиляция файлов

4. Проверка работы файлов. Открыла 3 терминала и запустила: ./server в одном и ./client в двух других. (Рис. 3.7).

В результате каждый терминал client выдал по 4 сообщения. Спустя менее 30 сек работа сервера прекратилась.

Работа выполнена корректно.



```
mrshcherbak@fedora:~  
[mrshcherbak@fedora ~]$ ./client  
FIFO Client...  
[mrshcherbak@fedora ~]$  
  
mrshcherbak@fedora:~  
[mrshcherbak@fedora ~]$ ./client  
FIFO Client...  
[mrshcherbak@fedora ~]$  
  
mrshcherbak@fedora:~  
[mrshcherbak@fedora ~]$ make all  
gcc client.c -o client  
[mrshcherbak@fedora ~]$ make all  
gcc server.c -o server  
[mrshcherbak@fedora ~]$ ./server  
FIFO Server...  
Wed Jun 1 14:37:58 2022  
Wed Jun 1 14:38:03 2022  
Wed Jun 1 14:38:07 2022  
Wed Jun 1 14:38:08 2022  
Wed Jun 1 14:38:12 2022  
Wed Jun 1 14:38:13 2022  
Wed Jun 1 14:38:17 2022  
Wed Jun 1 14:38:22 2022  
[mrshcherbak@fedora ~]$
```

Рис. 3.7: Выполнение

Если сервер завершит свою работу, не закрыв канал, то появится ошибка “Невозможно создать FIFO” при запуске сервера снова, тк один канал уже имеется.

4 *Контрольные вопросы:*

1). Именованные каналы отличаются от неименованных наличием идентификатора канала, который представлен как специальный файл (соответственно имя именованного канала – это имя файла). Поскольку файл находится на локальной файловой системе, данное IPC используется внутри одной системы.

2). Чтобы создать неименованный канал из командной строки нужно использовать символ |, служащий для объединения двух и более процессов: процесс_1 | процесс_2 | процесс_3...

3). Чтобы создать именованный канал из командной строки нужно использовать либо команду «mknod», либо команду «mkfifo».

4). Неименованный канал является средством взаимодействия между связанными процессами – родительским и дочерним. Родительский процесс создает канал при помощи системного вызова: «int pipe(int fd[2]);». Массив из двух целых чисел является выходным параметром этого системного вызова. Если вызов выполнен нормально, то этот массив содержит два файловых дескриптора. fd[0] является дескриптором для чтения из канала, fd[1] – дескриптором для записи в канал. Когда процесс порождает другой процесс, дескрипторы родительского процесса наследуются дочерним процессом, и, таким образом, прокладывается трубопровод между двумя процессами. Естественно, что один из процессов использует канал только для чтения, а другой – только для записи. Поэтому, если, например, через канал должны передаваться данные из родительского процесса в дочерний, родительский процесс сразу после запуска дочернего процесса закрывает дескриптор канала для чтения, а дочерний процесс закрывает дескриптор

для записи. Если нужен двунаправленный обмен данными между процессами, то родительский процесс создает два канала, один из которых используется для передачи данных в одну сторону, а другой – в другую.

5). Файлы именованных каналов создаются функцией `mkfifo()` или функцией `mknod()`:

«`int mkfifo(const char pathname, mode_t mode);`», где первый параметр – путь, где будет располагаться FIFO (имя файла, идентифицирующего канал), второй параметр определяет режим работы с FIFO (маска прав доступа к файлу), «`mknod (namefile, IFIFO | 0666, 0)`», где `namefile` – имя канала, `0666` – к каналу разрешен доступ на запись и на чтение любому запросившему процессу), «`int mknod(const char pathname, mode_t mode, dev_t dev);`». Функция `mkfifo()` создает канал и файл соответствующего типа. Если указанный файл канала уже существует, `mkfifo()` возвращает -1. После создания файла канала процессы, участвующие в обмене данными, должны открыть этот файл либо для записи, либо для чтения. 6). При чтении меньшего числа байтов, чем находится в канале или FIFO, возвращается требуемое число байтов, остаток сохраняется для последующих чтений. При чтении большего числа байтов, чем находится в канале или FIFO, возвращается доступное число байтов. Процесс, читающий из канала, должен соответствующим образом обработать ситуацию, когда прочитано меньше, чем заказано.

7). Запись числа байтов, меньшего емкости канала или FIFO, гарантированно атомарно. Это означает, что в случае, когда несколько процессов одновременно записывают в канал, порции данных от этих процессов не перемешиваются. При записи большего числа байтов, чем это позволяет канал или FIFO, вызов `write(2)` блокируется до освобождения требуемого места. При этом атомарность операции не гарантируется. Если процесс пытается записать данные в канал, не открытый ни одним процессом на чтение, процессу генерируется сигнал `SIGPIPE`, а вызов `write(2)` возвращает 0 с установкой ошибки (`errno=ERRPIPE`) (если процесс не установил обработки сигнала `SIGPIPE`, производится обработка по умолчанию – процесс завершается).

8). Количество процессов, которые могут параллельно присоединяться к любому концу канала, не ограничено. Однако если два или более процесса записывают в канал данные одновременно, каждый процесс за один раз может записать максимум PIPE BUF байтов данных. Предположим, процесс (назовем его А) пытается записать X байтов данных в канал, в котором имеется место для Y байтов данных. Если X больше, чем Y, только первые Y байтов данных записываются в канал, и процесс блокируется. Запускается другой процесс (например, В); в это время в канале появляется свободное пространство (благодаря третьему процессу, считывающему данные из канала). Процесс В записывает данные в канал. Затем, когда выполнение процесса А возобновляется, он записывает оставшиеся X-Y байтов данных в канал. В результате данные в канал записываются поочередно двумя процессами. Аналогичным образом, если два (или более) процесса одновременно попытаются прочитать данные из канала, может случиться так, что каждый из них прочитает только часть необходимых данных.

9). Функция write записывает байты count из буфера buffer в файл, связанный с handle. Операции write начинаются с текущей позиции указателя на файл (указатель ассоциирован с заданным файлом). Если файл открыт для добавления, операции выполняются в конец файла. После осуществления операций записи указатель на файл (если он есть) увеличивается на количество действительно записанных байтов. Функция write возвращает число действительно записанных байтов. Возвращаемое значение должно быть положительным, но меньше числа count (например, когда размер для записи count байтов выходит за пределы пространства на диске). Возвращаемое значение -1 указывает на ошибку; errno устанавливается в одно из следующих значений: EACCES – файл открыт для чтения или закрыт для записи, EBADF – неверный handle-р файла, ENOSPC – на устройстве нет свободного места. Единица в вызове функции write в программе server.c означает идентификатор (дескриптор потока) стандартного потока вывода.

10). Прототип функции strerror: «char * strerror(int errornum);». Функция strerror

интерпретирует номер ошибки, передаваемый в функцию в качестве аргумента `-errno`, в понятное для человека текстовое сообщение (строку). Откуда берутся эти ошибки? Ошибки эти возникают при вызове функций стандартных Си-библиотек. То есть хорошим тоном программирования будет – использование этой функции в паре с другой, и если возникнет ошибка, то пользователь или программист поймет, как исправить ошибку, прочитав сообщение функции `strerror`. Возвращенный указатель ссылается на статическую строку с ошибкой, которая не должна быть изменена программой. Дальнейшие вызовы функции `strerror` перезапишут содержание этой строки. Интерпретированные сообщения об ошибках могут различаться, это зависит от платформы и компилятора.

5 Выводы

Таким образом, в ходе ЛРН№14 я приобрела практические навыки работы с именованными каналами.